```
1
    # Importing required libraries
2
    from sqlalchemy import create engine, Column, Float, Integer, String
3
    from sqlalchemy.orm import sessionmaker, declarative_base
 4
    import sqlite3
5
    import pandas as pd
 6
    import numpy as np
7
    import matplotlib.pyplot as plt
    import os
8
9
    import pathlib
10
    import math
    import warnings
11
12
    import unittest
    from bokeh.plotting import figure, show
13
    from bokeh.io import output notebook
14
    from bokeh.models import ColumnDataSource
15
16
    import subprocess
    import os
17
    1 1 1
18
19
    SECTION 1.2
20
21
    # Suppressing warnings
22
    warnings.filterwarnings("ignore", category=UserWarning)
23
24
    # Defining the base class for SQLAlchemy
25
    Base = declarative_base()
26
    # Defining the TrainingData class
27
28
    class TrainingData(Base):
29
         __tablename__ = 'train_data'
30
        id = Column(Integer, primary_key=True)
        x = Column(Float)
31
        y1 = Column(Float)
32
        y2 = Column(Float)
33
        v3 = Column(Float)
34
35
        y4 = Column(Float)
36
    # Defining the DataVisualization class
37
38
    class DataVisualization:
        @staticmethod
39
40
        def plot_train_data_scatter():
41
42
             Plot scatter plot of training data.
43
44
            This function connects to the SQLite database, retrieves training data,
    and plots a scatter plot
45
             for visualizing the relationship between 'X' and 'Y1', 'Y2', 'Y3',
     'Y4'.
46
47
             Parameters:
```

```
48
             None
49
50
             Returns:
51
             None
             0.00
52
53
             try:
54
                 # Connecting to SQLite database
55
                 conn = sqlite3.connect('DataBase.db')
                 query = 'SELECT * FROM train_data'
56
57
                 df = pd.read_sql_query(query, conn)
                 x = df['x']
58
59
                 y1 = df['y1']
60
                 y2 = df['y2']
                 y3 = df['y3']
61
                 v4 = df['v4']
62
63
64
                 # Plotting scatter plot
65
                 plt.title('\n "X" against Y1, Y2, Y3, Y4 from trained data \n',
    fontdict={'fontsize': 20, 'fontweight': 5, 'color': 'Black'})
                 plt.xlabel("X values", fontdict={'fontsize': 10, 'fontweight': 10,
66
     'color': 'blue'})
                 plt.ylabel("Y Values", fontdict={'fontsize': 10, 'fontweight': 10,
67
     'color': 'blue'})
68
                 plt.scatter(x, y1, alpha=1, s=5, c='blue', label='X vs Y1')
69
                 plt.scatter(x, y2, alpha=1, s=5, c='red', label='X vs Y2')
70
                 plt.scatter(x, y3, alpha=1, s=5, c='green', label='X vs Y3')
71
72
                 plt.scatter(x, y4, alpha=1, s=5, c='yellow', label='X vs Y4')
73
74
                 plt.legend()
                 plt.show()
75
76
77
                 # Closing database connection
                 conn.close()
78
             except sqlite3.Error as e:
79
                 print(f"SQLite error: {e}")
80
             except Exception as e:
81
82
                 print(f"An error occurred: {e}")
83
    # Defining the DataProcessing class
84
85
    class DataProcessing:
        def find_sum_of_least_squares(self, x, y):
86
87
88
             Find the sum of least squares between two arrays.
89
90
             Parameters:
91
             x (numpy.array): First array.
92
             y (numpy.array): Second array.
93
94
             Returns:
95
             float: Sum of least squares.
```

```
96
 97
              try:
 98
                  x = np.array(x)
 99
                  y = np.array(y)
                  difference = np.subtract(x, y)
100
101
                  square = np.square(difference)
102
                  sum_of_squares = np.sum(square)
103
                  return sum_of_squares
104
              except Exception as e:
105
                  print(f"An error occurred: {e}")
106
107
          def find_least_squares(self, x, y):
108
109
              Find the least squares between two arrays.
110
              Parameters:
111
112
              x (numpy.array): First array.
113
              y (numpy.array): Second array.
114
115
              Returns:
              pandas.DataFrame: DataFrame containing least squares values.
116
117
118
              try:
119
                  x = np.array(x)
120
                  y = np.array(y)
                  difference = np.subtract(x, y)
121
                  square = np.square(difference)
122
123
                  return pd.DataFrame(square)
124
              except Exception as e:
125
                  print(f"An error occurred: {e}")
126
127
          def find_deviation(self, x, y):
128
129
              Find the deviation between two arrays.
130
              Parameters:
131
132
              x (numpy.array): First array.
133
              y (numpy.array): Second array.
134
135
              Returns:
136
              pandas.DataFrame: DataFrame containing deviation values.
              0.00
137
138
              try:
139
                  x = np.array(x)
140
                  y = np.array(y)
                  difference = np.subtract(x, y)
141
142
                  return pd.DataFrame(difference)
143
              except Exception as e:
144
                  print(f"An error occurred: {e}")
145
146
          def any_deviation_greater_than_threshold(self, x, y, threshold):
```

```
147
148
              Check if any deviation is greater than the threshold.
149
              Parameters:
150
              x (numpy.array): First array.
151
              y (numpy.array): Second array.
152
153
              threshold (float): Threshold value.
154
155
              Returns:
156
              bool: True if any deviation is greater than the threshold, False
     otherwise.
157
158
              trv:
159
                  x = np.array(x)
                  y = np.array(y)
160
                  difference = pd.DataFrame(np.subtract(x, y))
161
                  return (difference > threshold).any().any()
162
              except Exception as e:
163
                  print(f"An error occurred: {e}")
164
165
     # Defining the DataLoader class
166
167
     class DataLoader:
168
          def __init__(self, data_base, table_name, data_frame):
              self.Base = Base
169
              self.data_base = data_base
170
              self.table name = table name
171
              self.data frame = data frame
172
173
              self.engine = create_engine('sqlite:///{}.db'.format(self.data_base))
174
          def load_data(self):
175
176
177
              Load data into the SQLite database.
178
179
              Parameters:
180
              None
181
182
              Returns:
183
              None
              0.00
184
185
              try:
186
                  self.Base.metadata.create_all(self.engine)
187
                  table_name = self.table_name
188
                  self.data_frame.to_sql(table_name, con=self.engine,
     if_exists='replace', index=False)
              except Exception as e:
189
                  print(f"An error occurred: {e}")
190
191
          def close_connection(self):
192
193
194
              Close the SQLite database connection.
195
```

```
196
              Parameters:
197
              None
198
199
              Returns:
200
              None
              0.00
201
202
              trv:
203
                  self.engine.dispose()
204
              except Exception as e:
205
                  print(f"An error occurred: {e}")
206
207
      # Defining the Ideal class
208
      class Ideal(DataProcessing):
209
          def load_ideal_data(self):
210
              Load ideal data from the CSV file.
211
212
213
              Parameters:
              None
214
215
216
              Returns:
              pandas.DataFrame: Loaded ideal data.
217
218
219
              try:
                  self.ideal = pd.read_csv("Datasets/ideal.csv")
220
                  return self.ideal
221
222
              except FileNotFoundError as e:
223
                  print(f"File not found: {e}")
              except Exception as e:
224
                  print(f"An error occurred: {e}")
225
226
227
          def find_ideal(self, x):
228
229
              Find the ideal functions based on least squares.
230
              Parameters:
231
232
              x (numpy.array): Array for which ideal functions are calculated.
233
234
              Returns:
235
              pandas.DataFrame: DataFrame containing ideal functions.
236
237
              trv:
238
                  database = DataProcessing()
239
                  least_squares = [database.find_sum_of_least_squares(x,
      self.ideal.iloc[:, i]) for i in range(len(self.ideal.columns))]
240
                  first_four_least_squares = sorted(least_squares)[:4]
                  indices = [least_squares.index(i) for i in
241
      first_four_least_squares]
242
                  ideal_functions = [self.ideal.iloc[:, i] for i in indices]
243
                  ideal_functions = pd.DataFrame(ideal_functions).transpose()
244
                  ideal_functions.columns = ['y1', 'y2', 'y3', 'y4']
```

```
245
                  return ideal_functions
246
              except Exception as e:
247
                  print(f"An error occurred: {e}")
248
249
      # Defining the Test class
250
      class Test(DataProcessing):
251
          def load_test_data(self):
              . . .
252
253
              Load test data from the CSV file.
254
              Parameters:
255
256
              None
257
258
              Returns:
259
              pandas.DataFrame: Loaded test data.
260
261
              try:
262
                  self.test = pd.read csv("Datasets/test.csv")
                  return self.test
263
              except FileNotFoundError as e:
264
                  print(f"File not found: {e}")
265
266
              except Exception as e:
                  print(f"An error occurred: {e}")
267
268
      # Defining the Train class
269
      class Train(DataProcessing):
270
271
          def load_training_data(self):
272
              Load training data from the CSV file.
273
274
275
              Parameters:
276
              None
277
278
              Returns:
              pandas.DataFrame: Loaded training data.
279
              0.00\,\,0
280
281
              trv:
                  self.train = pd.read_csv("Datasets/train.csv")
282
                  return self.train
283
              except FileNotFoundError as e:
284
                  print(f"File not found: {e}")
285
              except Exception as e:
286
287
                  print(f"An error occurred: {e}")
288
289
          def get_deviation(self, x):
290
291
              Get deviation between x and y values in the training data.
292
293
              Parameters:
294
              x (numpy.array): Array for which deviation is calculated.
295
```

```
296
              Returns:
297
              pandas.DataFrame: DataFrame containing deviation values.
298
299
              try:
                  self.deviation = self.find_deviation(x, self.train.iloc[:, 1])
300
                  return self.deviation
301
302
              except Exception as e:
303
                  print(f"An error occurred: {e}")
304
     # Defining the TestYourCode class
305
306
     class TestYourCode(unittest.TestCase):
307
         def test_find_sum_of_least_squares(self):
308
              data_processing = DataProcessing()
309
              x = [1, 2, 3, 4, 5]
              y = [2, 4, 6, 8, 10]
310
              result = data_processing.find_sum_of_least_squares(x, y)
311
              self.assertEqual(result, 55)
312
313
         def test_find_least_squares(self):
314
              data_processing = DataProcessing()
315
              x = [1, 2, 3, 4, 5]
316
317
              y = [2, 4, 6, 8, 10]
              result = data_processing.find_least_squares(x, y)
318
              expected_result = pd.DataFrame([1, 4, 9, 16, 25], columns=['x'])
319
              pd.testing.assert_frame_equal(result, expected_result)
320
321
322
         def test find deviation(self):
323
              data_processing = DataProcessing()
              x = [1, 2, 3, 4, 5]
324
              y = [2, 4, 6, 8, 10]
325
              result = data_processing.find_deviation(x, y)
326
              expected_result = pd.DataFrame([-1, -2, -3, -4, -5], columns=['x'])
327
328
              pd.testing.assert_frame_equal(result, expected_result)
329
         def test_any_deviation_greater_than_threshold(self):
330
              data processing = DataProcessing()
331
              x = [1, 2, 3, 4, 5]
332
333
              y = [2, 4, 6, 8, 10]
              threshold = 5
334
              result = data_processing.any_deviation_greater_than_threshold(x, y,
335
     threshold)
              self.assertTrue(result)
336
337
338
     # Main block of code
     if __name__ = "__main__":
339
         try:
340
341
              # Defining file paths and names
              path = pathlib.Path(__file__).parent.resolve()
342
343
              dataset_path = os.path.join(path, "Datasets")
              ideal filename = "ideal"
344
              train_filename = "train"
345
```

```
346
              database name = "DataBase"
347
348
              # Reading and loading ideal data
              df_ideal = pd.read_csv(os.path.join(dataset_path, "
349
     {}.csv".format(ideal_filename)))
              ideal_table_name = ideal_filename + "_data"
350
351
              ideal_data_loader = DataLoader(database_name, ideal_table_name,
     df_ideal)
352
              ideal_data_loader.load_data()
              ideal_data_loader.close_connection()
353
354
355
              # Reading and loading training data
356
              df_train = pd.read_csv(os.path.join(dataset_path, "
     {}.csv".format(train filename)))
              train_table_name = train_filename + "_data"
357
              train_data_loader = DataLoader(database_name, train_table_name,
358
     df_train)
359
              train_data_loader.load_data()
              train_data_loader.close_connection()
360
361
              # Creating instances of classes
362
363
              train = Train()
364
              test = Test()
              ideal = Ideal()
365
              data_processing = DataProcessing()
366
367
368
              # Loading data
369
              train_data = train.load_training_data()
              test_data = test.load_test_data()
370
              ideal_data = ideal.load_ideal_data()
371
372
373
              # Finding ideal functions
374
              ideal_functions = ideal.find_ideal(train_data.iloc[:, 1])
              ideal_functions.insert(0, 'x', train_data.iloc[:, 0])
375
376
              # Calculating deviations
377
              deviation_between_training_and_ideal = pd.DataFrame([])
378
379
              for column in ideal_functions.columns:
                  deviation_between_training_and_ideal[column] =
380
     data_processing.find_deviation(train_data.iloc[:, 1], ideal_functions[column])
381
              deviation_between_training_and_ideal =
     pd.DataFrame(deviation_between_training_and_ideal)
382
383
              # Calculating absolute deviation
              absolute_deviation = deviation_between_training_and_ideal.abs()
384
              maximum_deviation = absolute_deviation.max().max()
385
386
              # Calculating sqrt(2) * maximum deviation
387
388
              sqrt_2 = math.sqrt(2)
389
              sqrt_2_maximum_deviation = sqrt_2 * maximum_deviation
390
```

```
391
              # Initializing arrays for x and y values
392
              x_values = np.array([])
393
             v1 values = np.array([])
             y2_values = np.array([])
394
             y3_values = np.array([])
395
396
             y4_values = np.array([])
397
398
             # Finding best fit values
             for t in test_data.iloc[:, 0]:
399
                  least_squares = np.array([(x - t) ** 2 for x in
400
     ideal_functions.iloc[:, 0]])
401
402
                  index = np.argmin(least_squares)
403
                  x values = np.append(x values, ideal functions.iloc[index, 0])
                  y1_values = np.append(y1_values, ideal_functions.iloc[index, 1])
404
                  y2_values = np.append(y2_values, ideal_functions.iloc[index, 2])
405
406
                  y3_values = np.append(y3_values, ideal_functions.iloc[index, 3])
407
                  y4_values = np.append(y4_values, ideal_functions.iloc[index, 4])
408
409
              # Creating DataFrame for best fit values
              best fit values = pd.DataFrame([x values, y1 values, y2 values,
410
     y3 values, y4 values]).transpose()
411
              best_fit_values.columns = ['x', 'y1', 'y2', 'y3', 'y4']
412
              # Creating DataVisualization object
413
              data visualization = DataVisualization()
414
415
416
              # Initializing lists for DataFrames
417
             y1 = pd.DataFrame([])
             y2 = pd.DataFrame([])
418
             y3 = pd.DataFrame([])
419
420
             y4 = pd.DataFrame([])
421
422
             table_list = [y1, y2, y3, y4]
423
              # Creating DataFrames for each ideal function
424
425
              for i in range(1, 5):
426
                  if data_processing.any_deviation_greater_than_threshold(
427
                          best_fit_values.iloc[:, i], test_data.iloc[:, 1],
     sqrt_2_maximum_deviation):
                      print('y' + str(i) + ' is not in range')
428
429
                  else:
430
                      table_list[i - 1]['x'] = test_data.iloc[:, 0]
431
                      table_list[i - 1]['y'] = test_data.iloc[:, 1]
                      table_list[i - 1]['delta'] =
432
     data_processing.find_deviation(test_data.iloc[:, 1],
433
     best_fit_values.iloc[:, i])
                      table_list[i - 1]['ideal function'] = best_fit_values.iloc[:,
434
     i]
435
```

```
# Loading test data into SQLite database
436
437
                      test_loader = DataLoader(database_name, 'test_' + 'y' + str(i +
     1), table_list[i - 1])
                      test_loader.load_data()
438
439
440
              # Loading training data and ideal functions into SQLite database
441
              training_loader = DataLoader(database_name, 'training_data',
     best fit values)
              training_loader.load_data()
442
443
              training_loader.close_connection()
444
445
              ideal_loader = DataLoader(database_name, 'ideal_functions', ideal_data)
446
              ideal_loader.load_data()
447
              ideal loader.close connection()
448
449
              # Plotting the scatter plot
450
              data_visualization.plot_train_data_scatter()
451
452
         except Exception as e:
              print(f"An error occurred: {e}")
453
454
455
456
      1.1.1
457
458
     SECTION 1.3
459
460
     # Set your GitHub username and repository name
     github_username = "Name22" #please enter your github username
461
462
     repository_name = "test"#please enter your repo name in github
     repository url = f"https://github.com/{github username}/{repository name}.git"
463
464
465
     # Set the directory where you want to initialize the repository
466
     repository directory = "C:\\Users\\Name\\Downloads\\Python Assignment-2\\Python
     Assignment\\Python Assignment\\Task 2 Assignment" #Refer to this to amend the
     directory on your local machine
467
468
     # Change to the repository directory
469
     os.chdir(repository_directory)
470
471
     # Initialize a new Git repository
472
     subprocess.run(["git", "init"])
473
474
     # Add all files to the repository
475
     subprocess.run(["git", "add", "."])
476
477
     # Commit the changes
     subprocess.run(["git", "commit", "-m", "Initial commit"])
478
479
480
     # Add the GitHub remote repository
     subprocess.run(["git", "remote", "add", "origin", repository_url])
481
482
```

```
# Push the changes to the 'develop' branch
483
     result_push = subprocess.run(["git", "push", "-u", "origin", "develop"])
484
485
     # Print the result of pushing changes
486
     print("Result of pushing changes:", result_push)
487
488
489
     # Print a message indicating success
     print("Changes pushed successfully.")
490
491
492
```